Khoa
CÔNG NGHỆ THÔNG TIN
ĐH Khoa học Tự nhiên TPHCM

Bài 08: Kiến trúc x86-32bit

Phạm Tuấn Sơn

ptson@fit.hcmus.edu.vn

Lịch sử phát triển vi xử lý Intel

- Intel 4004 (1971)
  - Vi xử lý đầu tiên của Intel
  - 4-bit
- Intel 8080 (1972)
  - Thanh ghi 8-bit
  - Đường truyền dữ liệu 8-bit
  - Đường truyền địa chỉ 16-bit (có thể truy xuất bộ nhớ RAM 64 KB)
  - Được sử dụng trên máy tính cá nhân đầu tiên - Altair
- Intel 8086/8088 (1978)
  - Thanh ghi 16-bit
  - Đường truyền dữ liệu 16-bit (8088: 8-bit)
  - Đường truyền địa chỉ 20-bit
  - Được dùng trên máy tính cá nhân IBM PC đầu tiên
- Intel 80286 (1982)
  - Có thể truy xuất bộ nhớ 16 MB
  - Đường truyền địa chỉ 24-bit

Lịch sử phát triển vi xử lý Intel (tt)

- Kiến trúc x86-32bit (IA-32)
  - Intel 80386/ i386 (1985)
    - Thanh ghi 32 bit
    - Đường truyền địa chỉ 32-bit
  - Intel 80486/ i486 (1989)
    - Kỹ thuật đường ống (pipelining)
  - Pentium (1993)
    - Đường truyền dữ liệu 64-bit
    - Siêu vô hướng (2 đường ống song song)
  - Pentium Pro (1995), II (1997), III (1999), IV (2000), M (2003).

Lịch sử phát triển vi xử lý Intel (tt)

- Kiến trúc x86-64bit
  - Athlon64 của AMD (2003)
    - Bộ vi xử lý x86-64bit đầu tiên
  - Pentium 4 Prescott (2004)
  - Core 2 (2006), Core i3, i5, i7, Atom (2008)
  - Intel Sandy Bridge (2010)
- Kiến trúc IA-64
  - Itanium (2001)

Kiến trúc x86-32bit

- Chế độ hoạt động
- Tổ chức bộ nhớ
- Tập thanh ghi
- Tập lệnh
- Ngăn xếp
- Thủ tục

Chế độ hoạt động

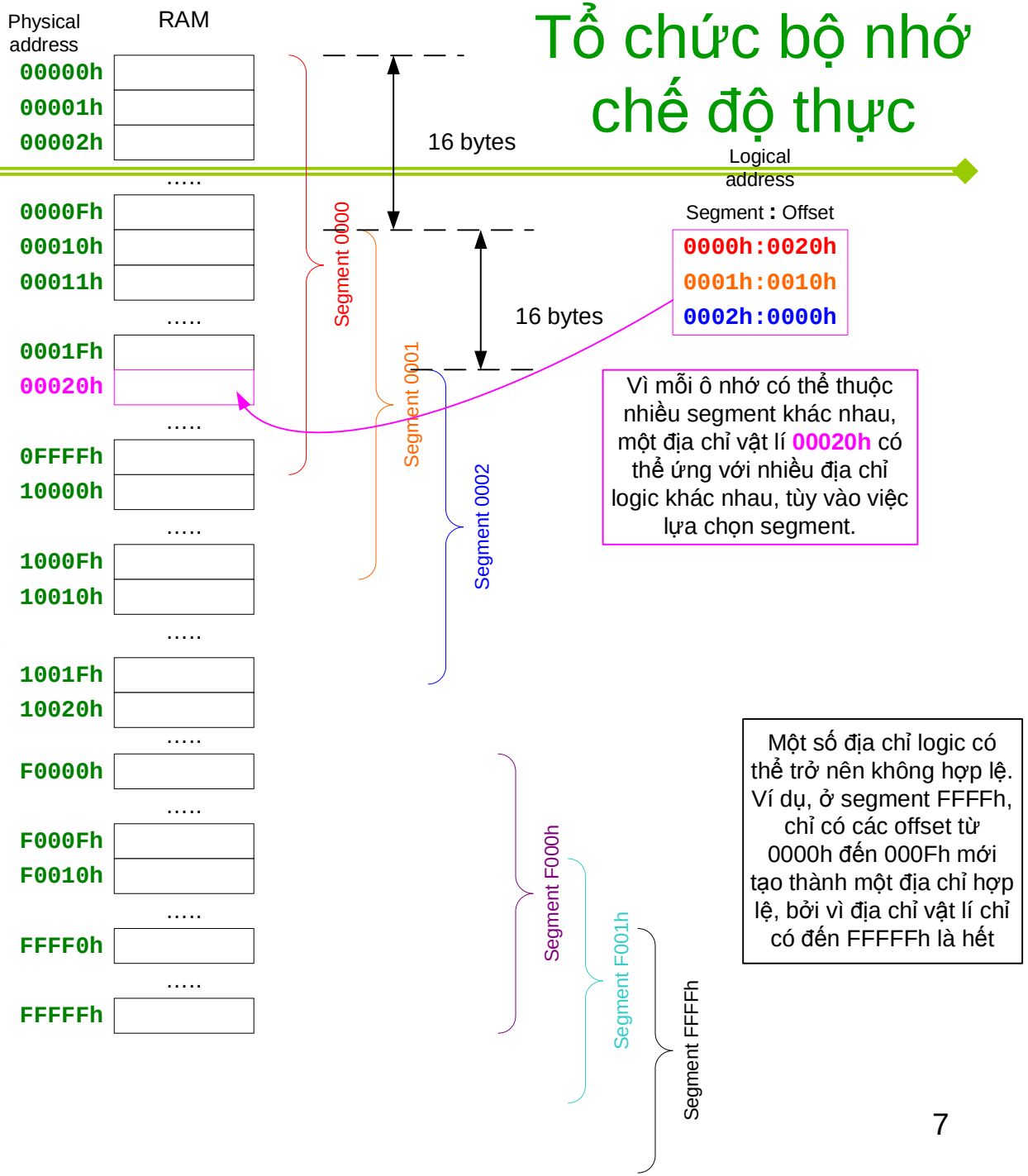
- Chế độ thực
  - 16 bit (8086)
  - Truy xuất 1 MB bộ nhớ chính
  - MS-DOS
- Chế độ bảo vệ
  - 32 bit
  - Truy xuất 4 GB bộ nhớ chính
  - Windows, Linux
  - Chế độ 8086 ảo
    - Chế độ thực dưới sự quản lý của chế độ bảo vệ
    - Cho phép hoạt động đồng thời ở 2 chế độ
- Chế độ quản lý hệ thống
  - Quản lý nguồn cung cấp
  - Chẩn lỗi và bảo mật hệ thống

Tổ chức bộ nhớ chế độ thực

Tại sao kích thước mỗi đoạn là 64 KB ?

Tại sao các đoạn lại cách nhau 16 byte ?

Tại sao các đoạn lại nằm chồng lên nhau ?



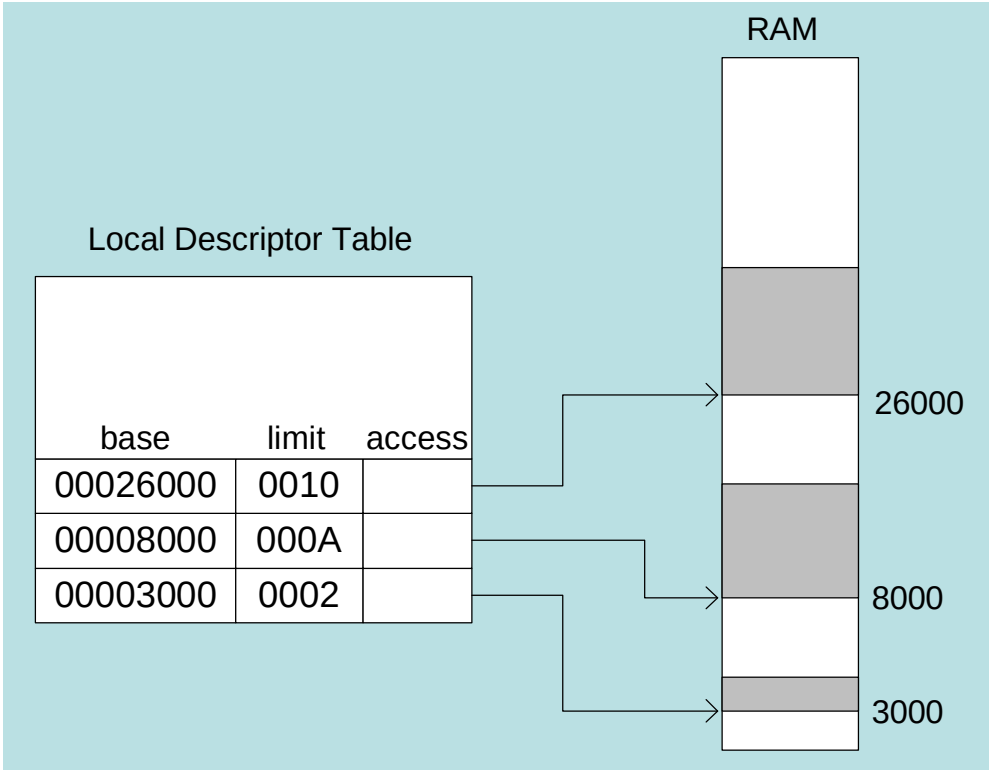
Chuyển đổi địa chỉ ở chế độ thực

- Địa chỉ logic **à** địa chỉ vật lý
  - Phy_address = segment * 10h + offset
  - Vd: địa chỉ logic 1234h:0005h sẽ ứng với địa chỉ vật lí
    1234h * 10h + 0005h = 12340h + 0005h = 12345h
- Địa chỉ vật lý **à** địa chỉ logic
  - Do các đoạn gối đầu nhau nên mỗi ô nhớ có thể thuộc một vài đoạn khác nhau. Vì vậy, một địa chỉ vật lý có thể ứng với nhiều địa chỉ logic khác nhau.
  - Vd: địa chỉ vật lý 12345h có thể ứng với các địa chỉ logic sau:

    1234h:0005h, 1230h:0045h
    1200h:0345h, 1000h:2345h
    1232h:0025h, …

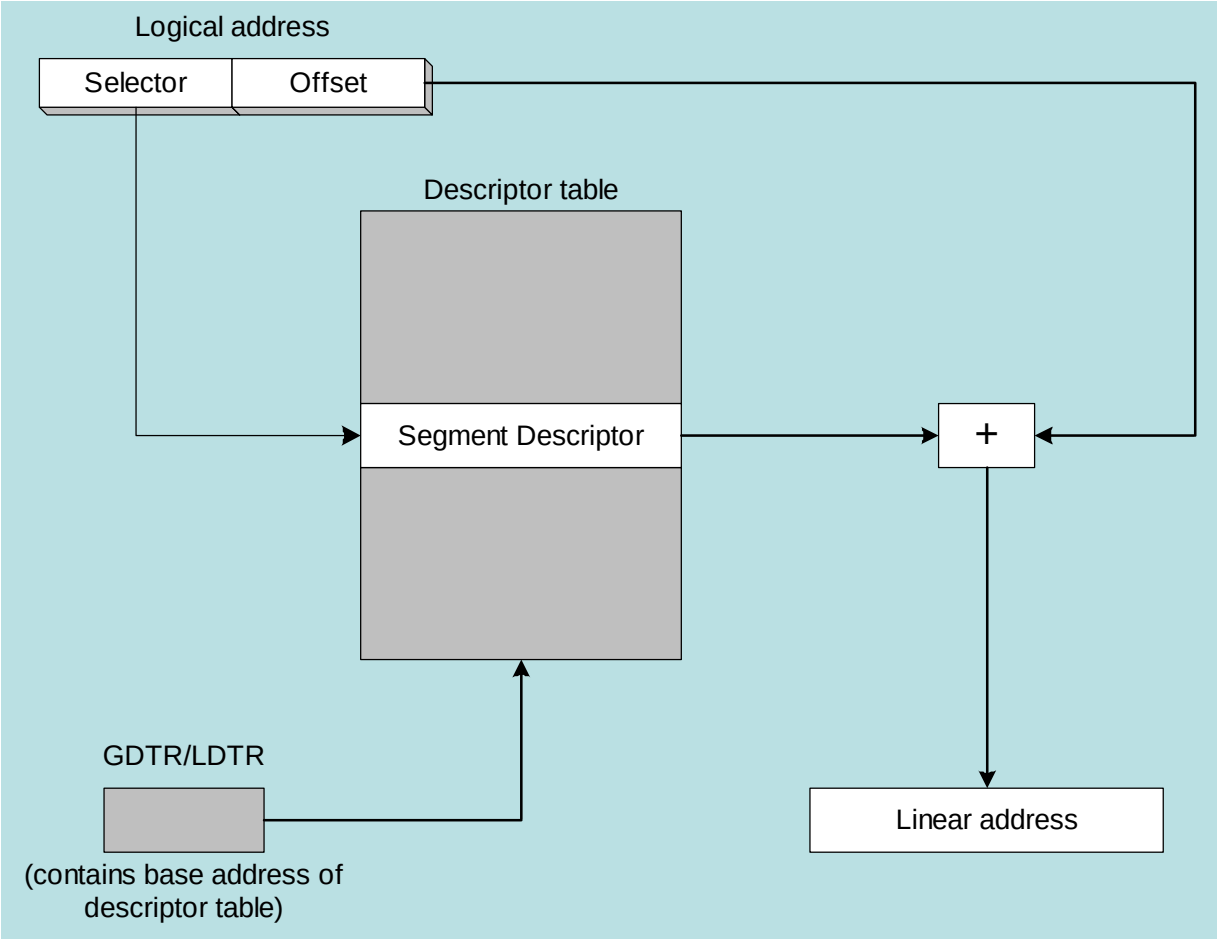
Tổ chức bộ nhớ chế độ bảo vệ

- Bộ nhớ cũng được chia thành các đoạn. Tuy nhiên, kích thước các đoạn không được định sẵn như chế độ thực.
- Do đó, để định vị một đoạn nào đó thì phải sử dụng một bảng mô tả các đoạn.
- Để truy xuất vào một ô nhớ trong bộ nhớ chính thì cũng phải thực hiện chuyển đổi từ địa chỉ logic (segment, offset) thành địa chỉ vật lý



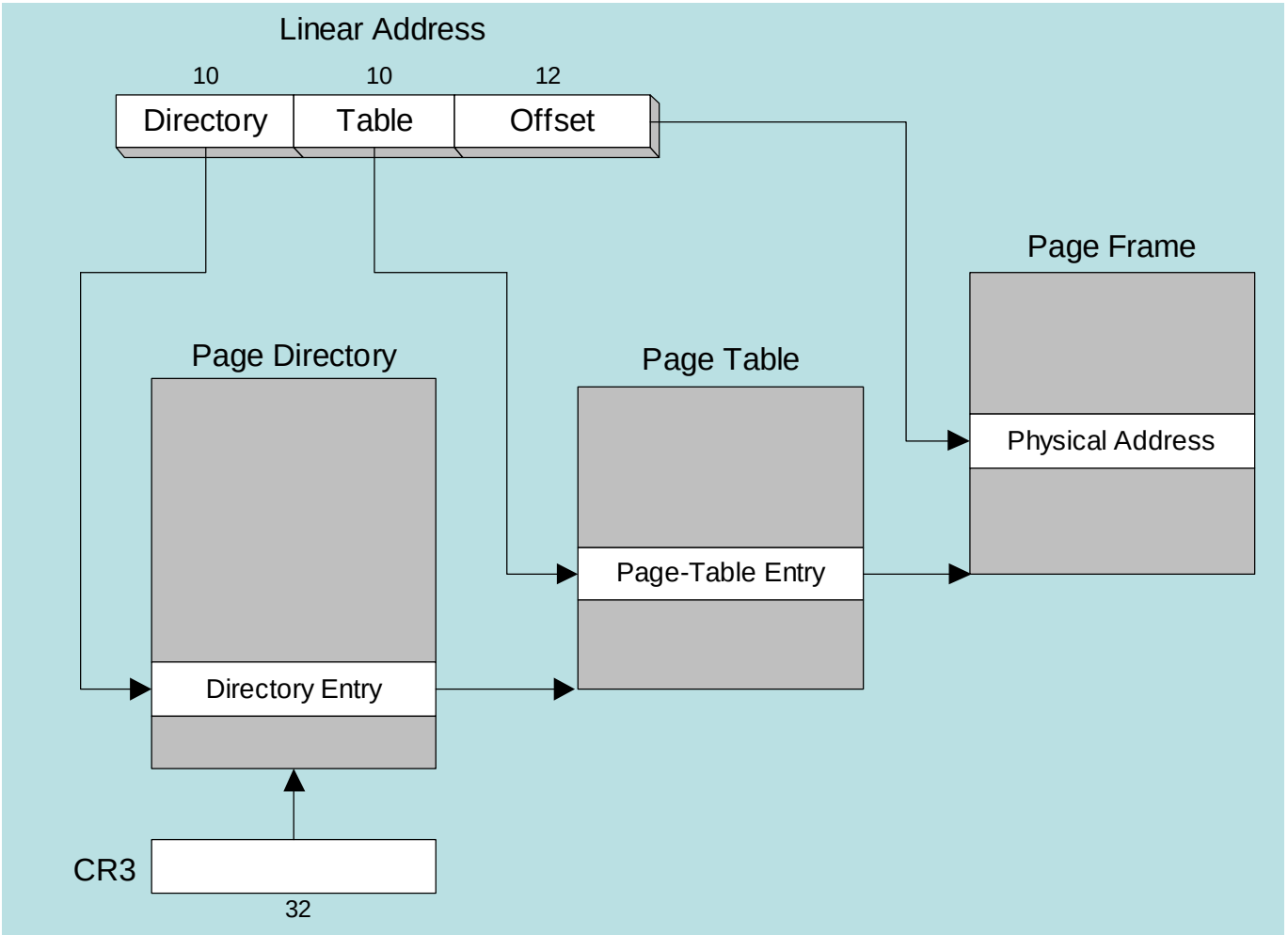
Chuyển đổi địa chỉ ở chế độ bảo vệ

- Thực hiện quá trình chuyển đổi địa chỉ một bước hoặc hai bước để chuyển đổi từ địa chỉ logic (segment, offset) thành địa chỉ vật lý
- Bước 1, kết hợp segment và offset thành địa chỉ tuyến tính (linear address)



Chuyển đổi địa chỉ ở chế độ bảo vệ (tt)

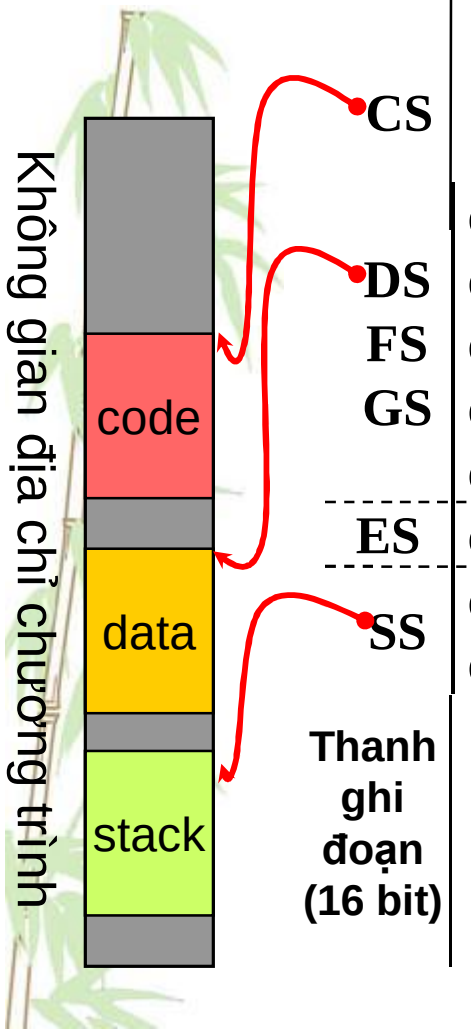
- Bước 2, chuyển địa chỉ tuyến tính thành địa chỉ vật lý (physical address)



Chương trình chạy trên hệ thống

- Chương trình chạy trên hệ thống *thông thường* chiếm 3 đoạn bộ nhớ
  - Một đoạn dành cho mã lệnh (code segment)
  - Một đoạn dành cho dữ liệu (data segment)
  - Một đoạn ngăn xếp (stack segment) dành để lưu các giá trị trung gian hoặc các địa chỉ trở về dùng khi gọi hàm
- Trên hệ thống x86, cần có các thanh ghi chứa địa chỉ đoạn và địa chỉ ô để truy xuất bộ nhớ

Tập thanh ghi



CS
DS
FS
GS
ES
SS

Thanh ghi đoạn (16 bit)

Các thanh ghi đa dụng (32 bit)

	31 16	15 8	7 0	bit
eax		ah	al	Tích lũy
ebx		bh	bl	Chỉ mục cơ sở
ecx		ch	cl	Đếm
edx		dh	dl	Dữ liệu
esi		si		Chỉ mục nguồn
edi		di		Chỉ mục đích
esp		sp		Con trỏ ngăn xếp
ebp		bp		Con trỏ cơ sở

Một số thanh ghi khác

- Thanh ghi chứa địa chỉ lệnh (EIP – 32 bit), kết hợp thanh ghi đoạn CS – 16bit (CS:EIP)
- Thanh ghi cờ (EFLAGS – 32 bit)

32	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
…….	VM	RF		NT	IO PL	IO PL	OF	DF	IF	TF	SF	ZF		AF		PF		CF

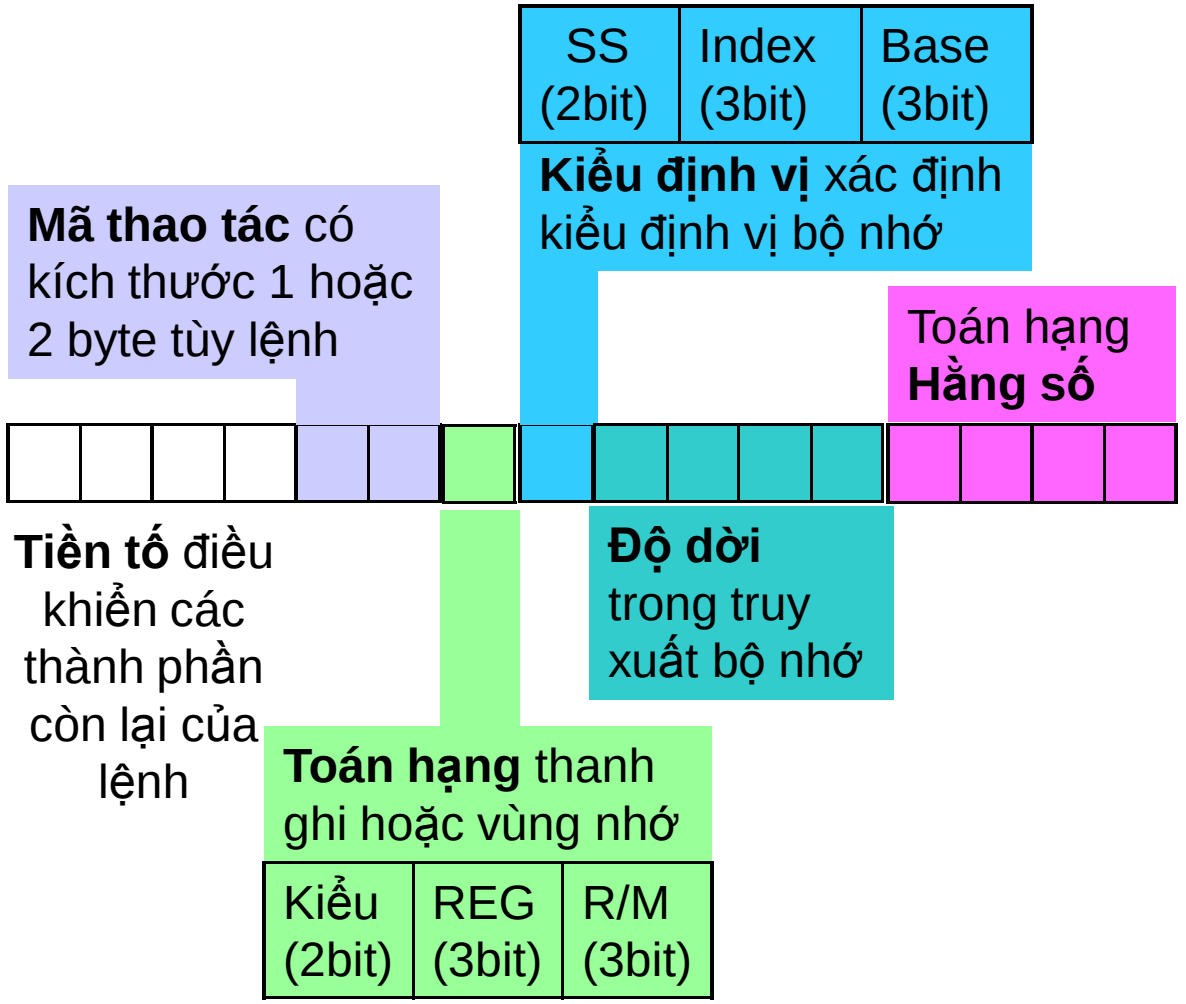
  - Carry: cờ tràn không dấu
  - Overflow: cờ tràn có dấu
  - Sign: cờ dấu
  - Zero: cờ zero
  - Auxiliary Carry: cờ nhớ từ bit 3 vào bit 4
  - Parity: cờ chẵn lẻ
  - …

  Giá trị của từng cờ được thiết lập sau mỗi lệnh được thực thi

- Một số thanh ghi khác: IDTR (16bit), GDTR (48bit), LDTR (48bit), TR (16bit), ...

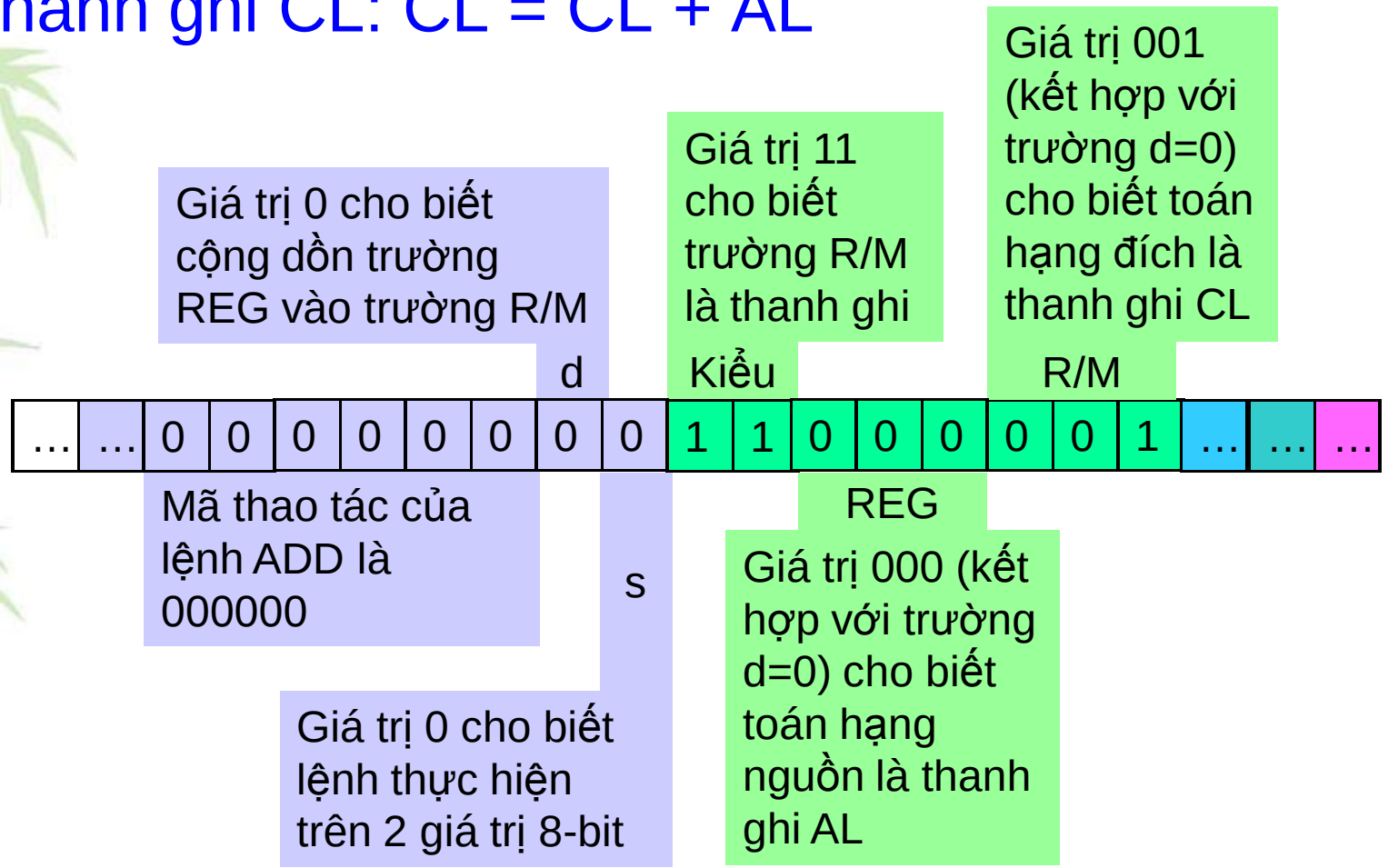
Cấu trúc lệnh

- Mặc dù trong cấu trúc lệnh có tổng cộng 16 byte nhưng thực tế chỉ cho phép lệnh dài tối đa 15 byte



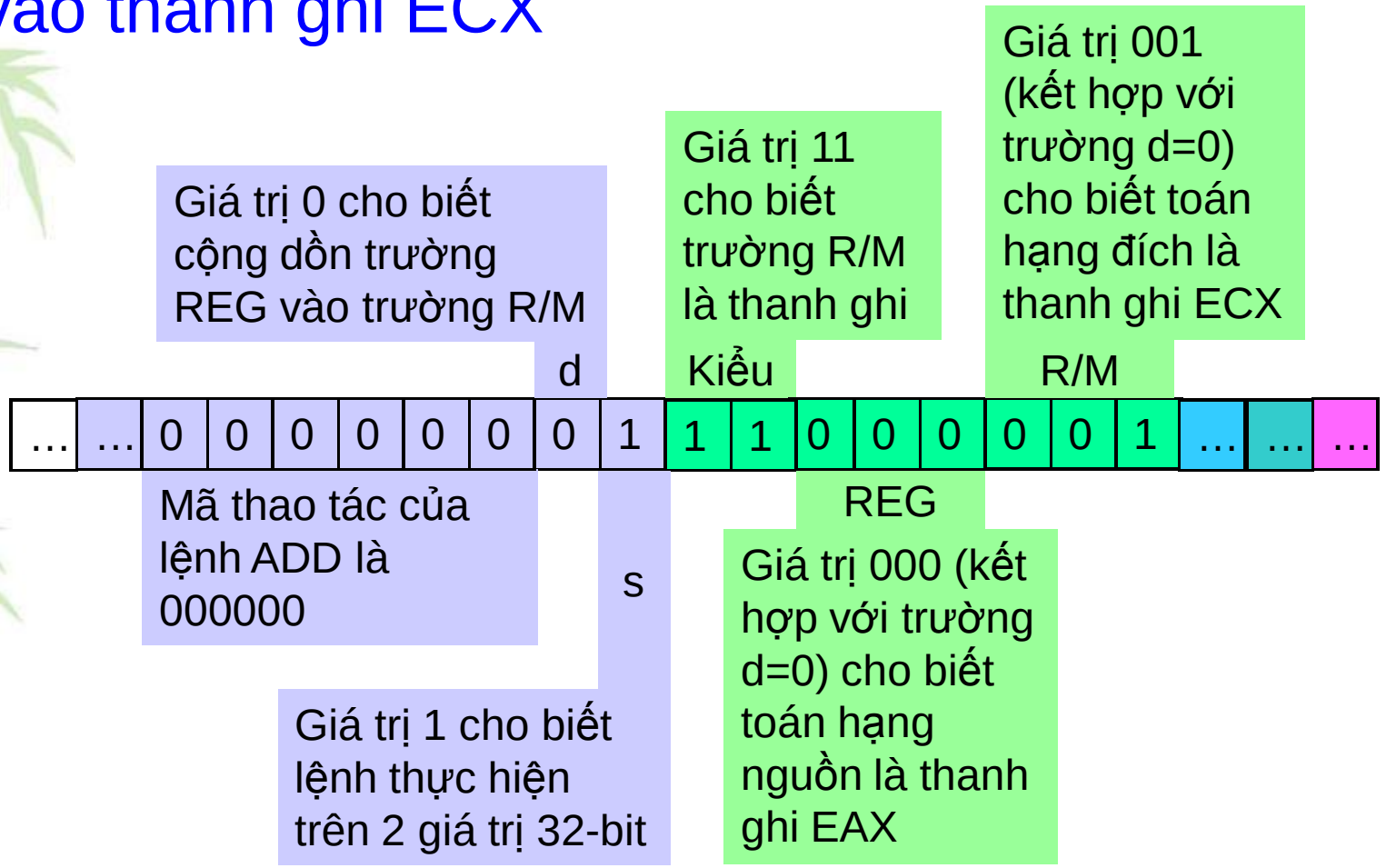
ADD CL, AL

- Lệnh này cộng dồn giá trị trong thanh ghi AL vào thanh ghi CL: CL = CL + AL



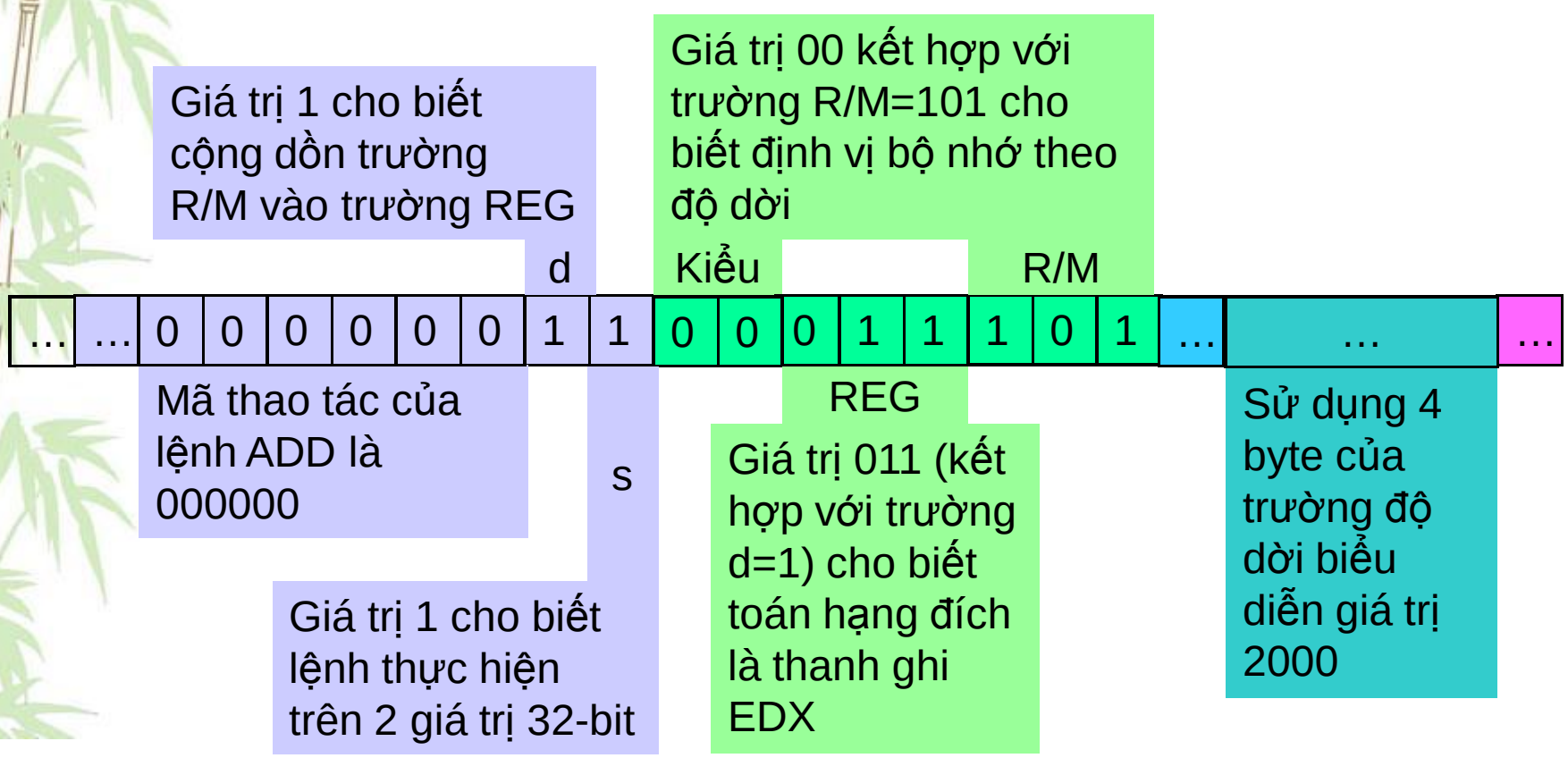
ADD ECX, EAX

- Lệnh này cộng dồn giá trị trong thanh ghi EAX vào thanh ghi ECX



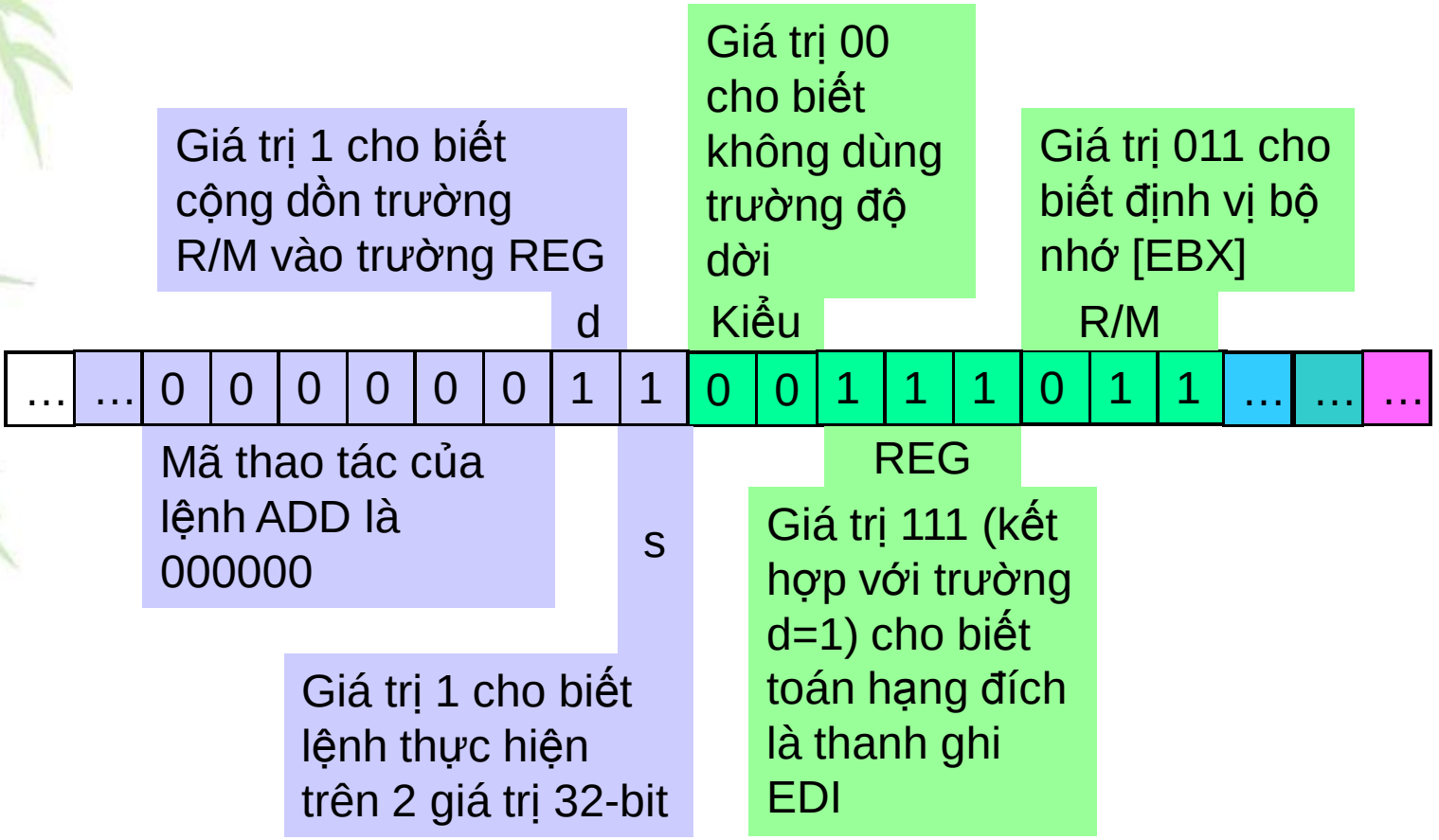
ADD EDX, [2000]

- Lệnh này cộng dồn giá trị từ nhớ 4 byte có địa chỉ bắt đầu là DS:2000 vào thanh ghi EDX



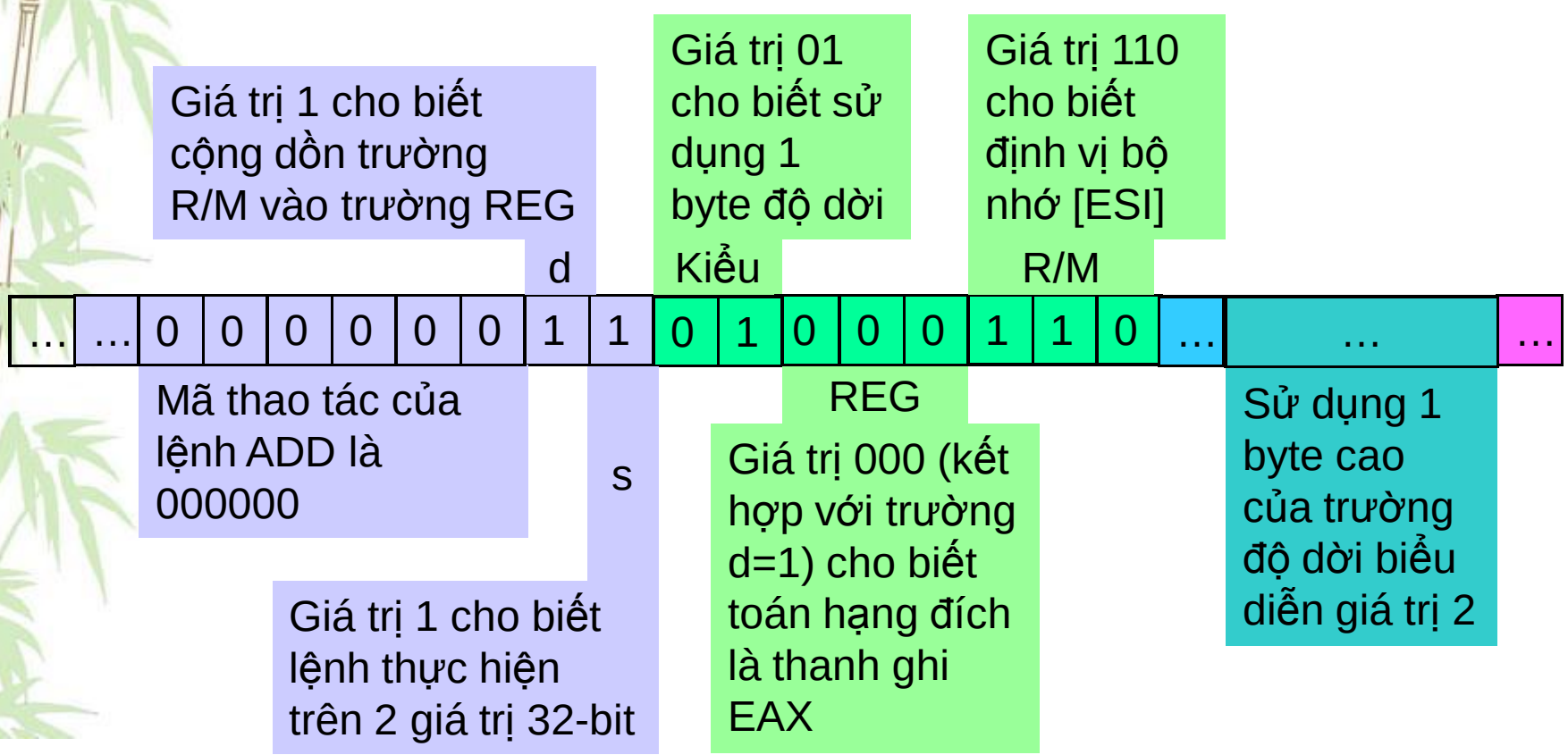
ADD EDI, [EBX]

- Lệnh này cộng dồn giá trị từ nhớ 4 byte có địa chỉ bắt đầu là DS:EBX vào thanh ghi EDI



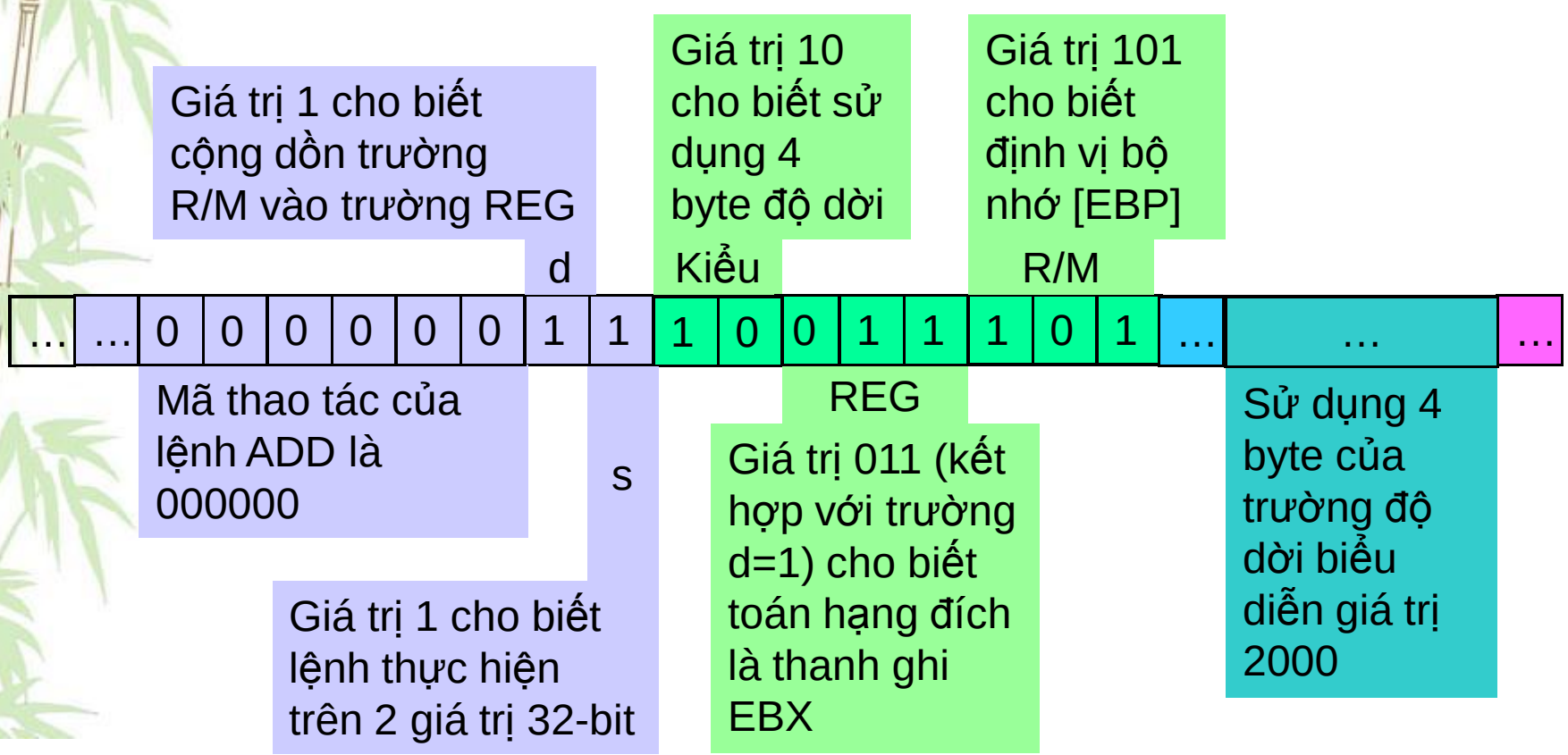
ADD EAX, [ESI + 2]

- Lệnh này cộng dồn giá trị từ nhớ 4 byte có địa chỉ bắt đầu là DS:(ESI+2) vào thanh ghi EAX



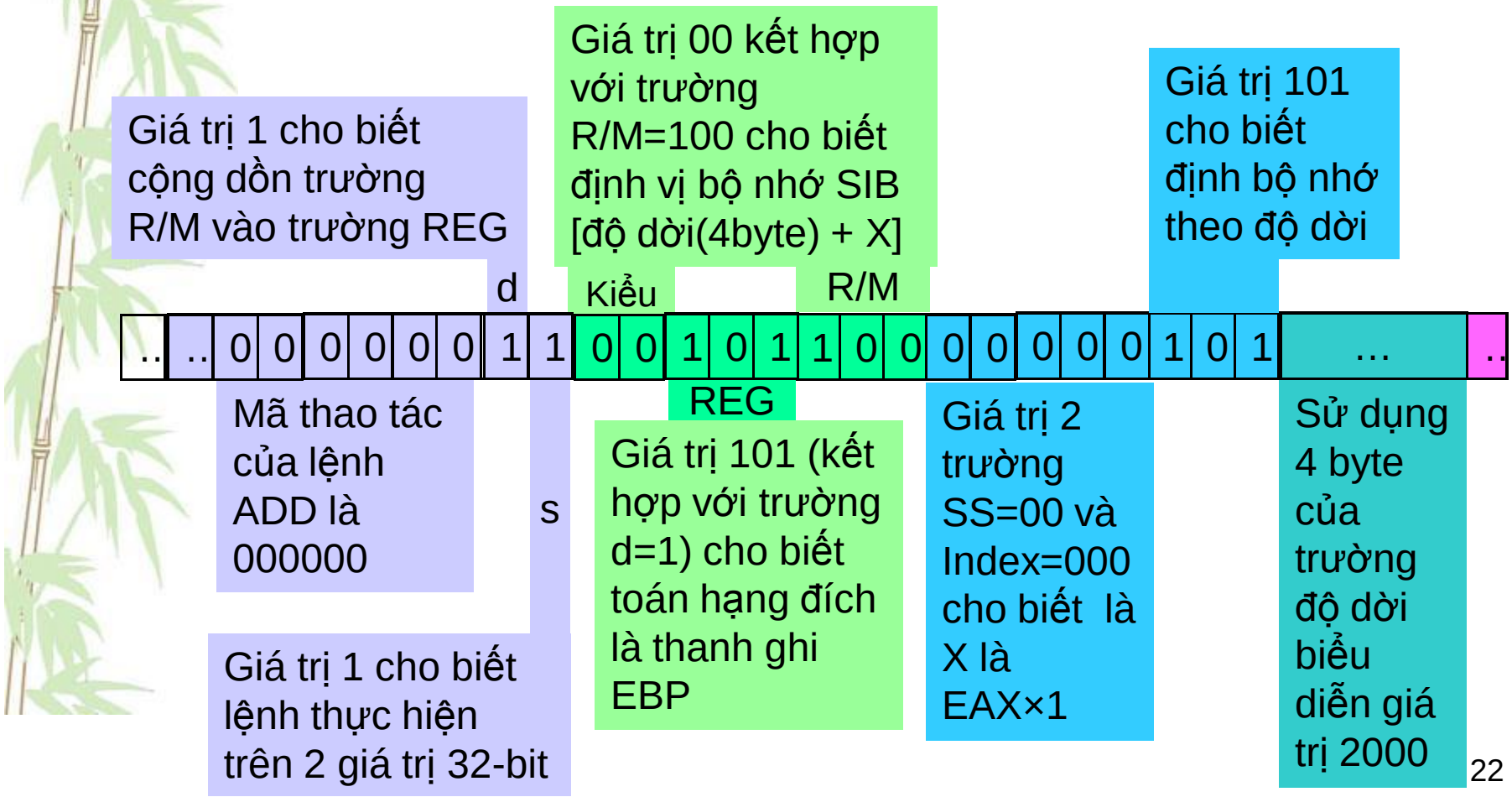
ADD EBX, [EBP + 2000]

- Lệnh này cộng dồn giá trị từ nhớ 4 byte có địa chỉ bắt đầu là SS:(EBP+2000) vào thanh ghi EBX



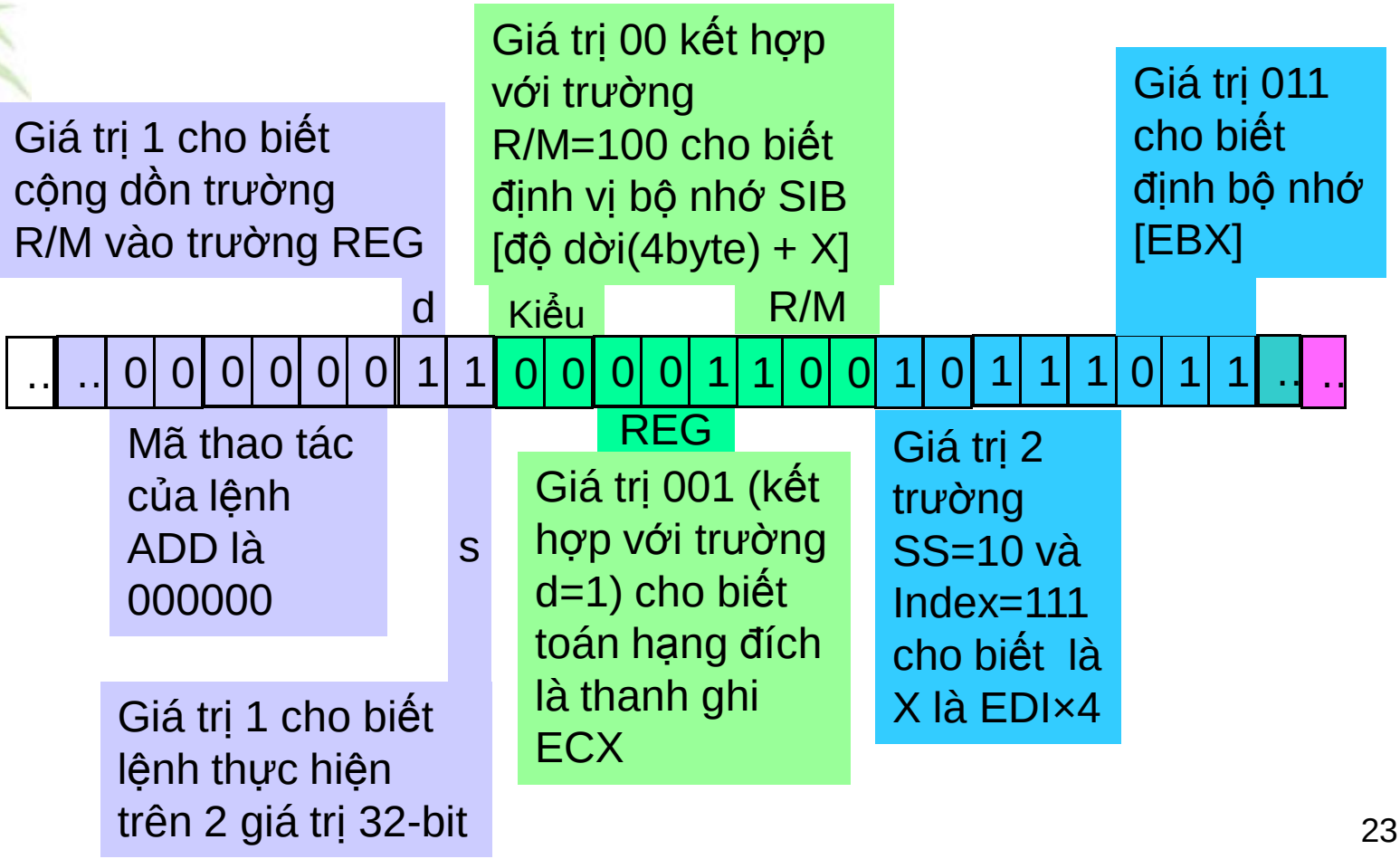
ADD EBP, [2000 + EAX×1]

- Lệnh này cộng dồn giá trị từ nhớ 4 byte có địa chỉ bắt đầu là DS:(EAX×1 + 2000) vào thanh ghi EBP



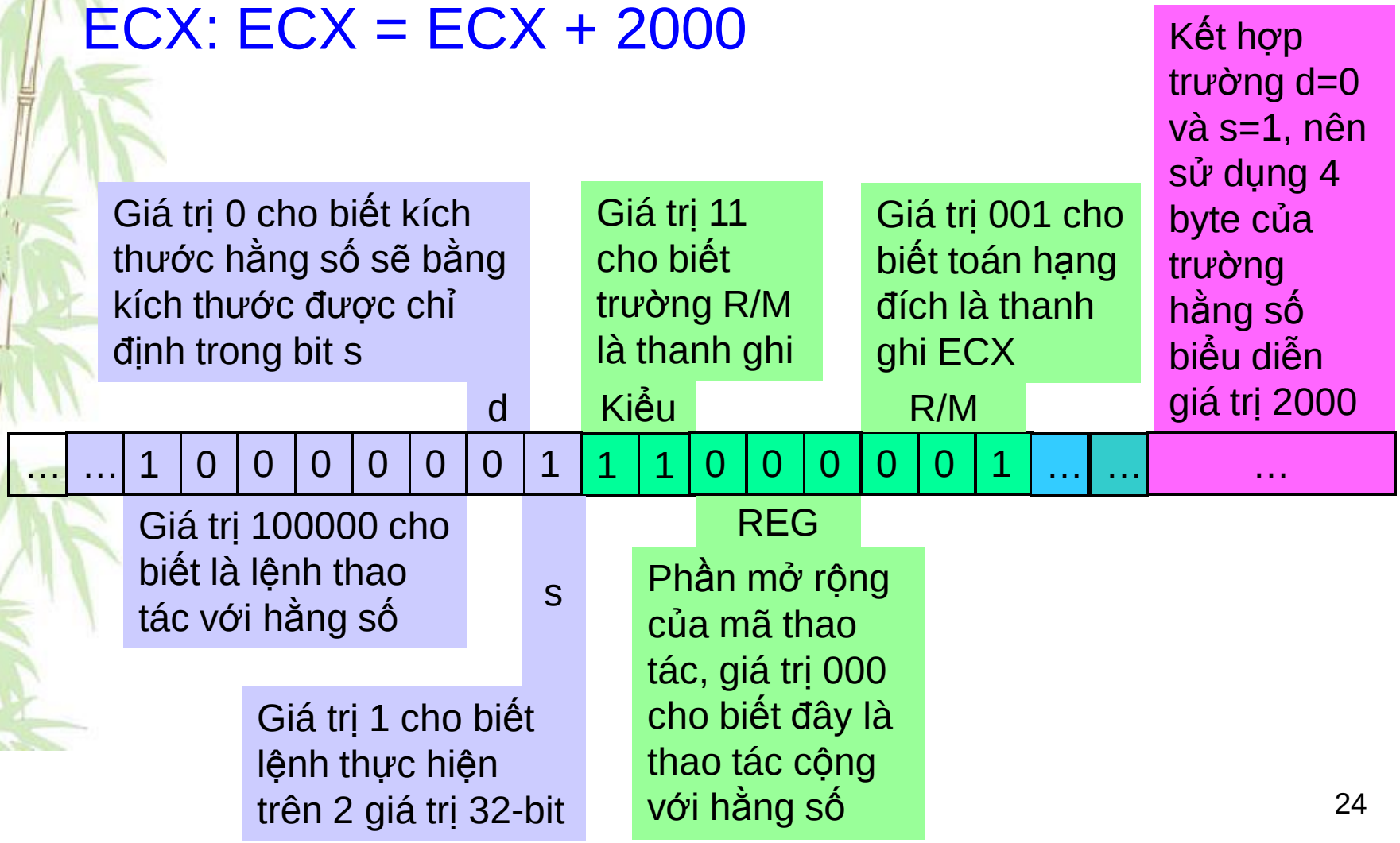
ADD ECX, [EBX + EDI×4]

- Lệnh này cộng dồn giá trị từ nhớ 4 byte có địa chỉ bắt đầu là DS:(EDI×4 + EBX) vào thanh ghi ECX



ADD ECX, 2000

- Lệnh này cộng dồn giá trị 2000 vào thanh ghi ECX: ECX = ECX + 2000



So sánh lệnh MIPS và x86 32 bit (1/3)

- MIPS: "Kiến trúc 3 toán hạng"
  - 2 toán hạng nguồn và một toán hạng đích

```
add $s0,$s1,$s2 # s0=s1+s2
```

  - Ưu điểm: ít lệnh hơn ⇒ Tốc độ xử lý nhanh hơn

- x86: "Kiến trúc 2 toán hạng"
  - 1 toán hạng nguồn và 1 toán hạng đóng 2 vai trò toán hạng đích và toán hạng nguồn

```
add EBX,EAX ; EBX=EBX+EAX
```

  - Ưu điểm: lệnh ngắn hơn ⇒ Mã nguồn nhỏ hơn

So sánh lệnh MIPS và x86 32 bit (2/3)

- MIPS: "Kiến trúc nạp-lưu"
  - Chỉ có lệnh Load/Store truy xuất bộ nhớ;các lệnh còn lại thao tác trên thanh ghi, hằng số

```
lw $t0, 12($gp)
add $s0, $s0,$t0 # s0=s0+Mem[12+gp]
```

  - Ưu điểm: Mạch xử lý đơn giản hơn ⇒ dễ nâng cao tốc độ bằng cách sử dụng các kỹ thuật song song hóa

- x86: "Kiến trúc thanh ghi - bộ nhớ"
  - Tất cả các lệnh đều có thể truy xuất bộ nhớ

```
ADD EAX,[ESI + 12] ;EAX=EAX+Mem[12+ESI]
```

  - Ưu điểm: ít lệnh hơn ⇒ mã nguồn nhỏ hơn

So sánh lệnh MIPS và x86 32 bit (3/3)

- MIPS: "Các lệnh có chiều dài cố định"
  - Tất cả các lệnh đều có kích thước 4 byte
  - Mạch xử lý đơn giản hơn ⇒ Tốc độ xử lý nhanh hơn
  - Lệnh nhảy: bội số của 4 byte

- x86: "Các lệnh có chiều dài thay đổi"
  - Lệnh có kích thước thay đổi từ 1 byte tới 16 byte

  ⇒ Kích thước mã nguồn có thể nhỏ hơn (30% ?)
  - Sử dụng bộ nhớ cache hiệu quả hơn
  - Lệnh có thể có hằng số 8 bit hoặc 32 bit

Cú pháp lệnh hợp ngữ x86-32bit

- Hợp ngữ trên x86 có 2 cú pháp chính
  - **Cú pháp Intel**
  - Cú pháp AT&T
- <Tên lệnh> <toán hạng đích>, <toán hạng nguồn>

  ; Ghi chú
- Các loại toán hạng
  - Thanh ghi: EAX, AX, AL,…
  - Ô nhớ: [EBX], [EBX+ESI+7], biến, …
  - Hằng số: 5, -24, 3Fh, 10001101b…
- Ví dụ

  mov EAX, 2000 ; gán thanh ghi EAX = 2000

Các lệnh tính toán, tương tác bộ nhớ

- add…
- sub…
- nd…,or…, xor…
- nor
- sll, srl, sra
- lw
- sw
- lb
- lh
- add/addi
- lui
- x

- add
- sub
- and, or, xor
- not
- sal, shr, sar
- mov reg, mem
- mov mem, reg
- BYTE PTR
- WORD PTR
- mov reg, reg/imm
- x
- lea

```
lea esi, -4000000(ebp)
# esi = ebp - 4000000
```

Ví dụ

```
section     .data
        a       DW 4321h
                DW 8765h
        b       DW 0FFFFh
                DW 0

section     .code

        ; perform b = b + a
        MOV     AX, a
        MOV     BX, a+2
        ADD     b, AX     ; 4320h with CF=1
        ADC     b+2,BX    ; 8766h

        ; perform b = b + BX:AX
        MOV     AX, 5320h
        MOV     BX, 0
        SUB     b, AX     ; F000h with CF=1
        SBB     b+2, BX   ; 8765h
```

```
        MOV     CX, 128

        ; perform DX:AX = AX * CX
        MOV     AX, 0F000h  ; 61440 dec
        MUL     CX          ; DX:AX = 0078:0000
                            ; (7864320=61440*128)

        MOV     AX, 0F000h  ; -4096 dec
        IMUL    CX          ; DX:AX = FFF8:0000
                            ; (-524288=-4096*128)

        ; perform AX = DX:AX / CX
        MOV     AX, 0F000h          ; 61440 dec
        DIV     CX                  ; AX = 01E0h

        MOV     AX, 0F000h          ; -4096 dec ???
        IDIV    CX                  ; AX = ?

        MOV     AX, 0F000h      ; -4096 dec
        CWD
        IDIV    CX          ; AX = FFE0h = -32

        NEG     AX          ; 0020h = 32
```

Ví dụ

```
section     .code

    MOV     AL, 36h
    AND     AL, 0Fh                     ; AL = 06h
                                        ; AL = 00000110b
    AND     AL, 00000010b               ; AL = 00000010b = 02h
    OR      AL, 30h                     ; AL = 32h
    XOR     AL, AL                      ; AL = 0
    NOT     AL                          ; AL = FFh
    MOV     AX, 1234h
    MOV   CL, 4
    SHR    AX, CL          ; 0123h
    SHL    AX, CL          ; 1230h

    MOV    AL, -4             ; -4 = FCh = 11111100
    SAR    AL, 1            ; -2 = FEh = 11111110

    MOV    AL, -4             ;  -4 = FCh = 11111100
    SHR    AL, 1            ; 126 = 7Eh = 01111110

    MOV    AL, 10101010b    ; AAh
    ROL    AL, 1            ; 01010101 = 55h

    MOV    AL, 10101010b
    STC                       ; CF = 1
    RCR    AL, 1            ; 11010101 = D5h  CF = 0
```

Nhảy & lặp

- JMP
- J<cc>
  - Nhảy theo cờ với kết quả không dấu
    - JA(JNBE), JB(JNAE), JE(JZ), JNA(JBE), JNB(JAE), JNE(JNZ)
  - Nhảy theo cờ với kết quả có dấu
    - JG(JNLE), JL(JNGE), JE(JZ), JNG(JLE), JNL(JGE), JNE(JNZ)
  - Nhảy theo trị của một cờ
    - JC, JZ(JE), JS, JO, JNC, JNZ(JNE), JNS, JNO
- JCXZ
- LOOP
- LOOPE / LOOPNE, LOOPZ / LOOPNZ

So sánh lệnh nhảy trong MIPS và x86-32bit

- So sánh các thanh ghi và thực hiện nhảy dựa vào kết quả so sánh
  - `beq`
  - `bne`
  - `slt; beq`
  - `slt; bne`
- Thực hiện nhảy dựa vào giá trị các cờ S, Z, C, O,… Thường sử dụng lệnh cmp trước các lệnh nhảy để thay đổi giá trị các cờ
  - `(cmp;) je`
  - `(cmp;) jne`
  - `(cmp;) jlt`
  - `(cmp;) jge`

Rẽ nhánh (1/4)

Trong ngôn ngữ C

```
If (AX==0)
        AX = AX + 1;
BX = AX;

If (AX<0)
        AX = AX + 1;
Else
        AX = AX – 1;
BX = AX;
```

Trong ASM (C2)

```
    CMP AX, 0
    JNE TIEP
    INC AX
TIEP:
    MOV BX, AX

    CMP AX, 0
    JNL LONHON
    INC AX
    JMP TIEP
LONHON:
    DEC AX
TIEP:
    MOV BX, AX
```

Trong ASM (C1)

```
    CMP AX, 0
    JE CONG
    JMP TIEP
CONG:
    INC AX
TIEP:
    MOV BX, AX

    CMP AX, 0
    JL NHOHON
    DEC AX
    JMP TIEP
NHOHON:
    INC AX
TIEP:
    MOV BX, AX
```

Rẽ nhánh (2/4)

Trong ngôn ngữ C

```
If (AL==‘S’)
        printf (“Chao buoi sang”);
else if (AL==‘T’)
        printf (“Chao buoi trua”);
else if (AL==‘C’)
        printf (“Chao buoi chiều”);
```

Trong ASM (C2)

```
    CMP AL, 'S'
    JNE KP_SANG
    ; xuất thông báo
    ; “Chao buoi sang”
    ;
    JMP THOAT
KP_SANG:
    CMP AL, 'T'
    JNE KP_TRUA
    ; xuất thông báo
    ; “Chao buoi trua”
    ;
    JMP THOAT
KP_TRUA:
    CMP AL, 'C'
    JNE THOAT
    ; xuất thông báo
    ; “Chao buoi chieu”
    ;
THOAT:
```

Trong ASM (C1)

```
    CMP AL, 'S'
    JE CHAO_BUOI_SANG
    CMP AL, 'T'
    JE CHAO_BUOI_TRUA
    CMP AL, 'C'
    JE CHAO_BUOI_CHIEU
    JMP THOAT
CHAO_BUOI_SANG:
    ; xuất thông báo
    ; “Chao buoi sang”
    ;
    JMP THOAT
CHAO_BUOI_TRUA:
    ; xuất thông báo
    ; “Chao buoi trua”
    ;
    JMP THOAT
CHAO_BUOI_CHIEU:
    ; xuất thông báo
    ; “Chao buoi chieu”
    ;
THOAT:
```

Rẽ nhánh (3/4)

Trong ngôn ngữ C	Trong ASM (C2)	Trong ASM (C1)	Trong ASM (C3)
If (AL>='a' and AL<='z')	CMP AL, 'a'	CMP AL, 'a'	CMP AL, 'a'
	JB KPTHUONG	JAE THUONG	JB KPTHUONG
	CMP AL, 'z'	CMP AL, 'z'	CMP AL, 'z'
	JA KPTHUONG	JBE THUONG	JBE THUONG
AX = AX + 1;	INC AX	DEC AX	KPTHUONG:
	JMP TIEP	JMP TIEP	DEC AX
else	KPTHUONG:	THUONG:	JMP TIEP
AX = AX – 1;	DEC AX	INC AX	THUONG:
	TIEP:	TIEP:	INC AX
BX = AX;	MOV BX, AX	MOV BX, AX	TIEP:
			MOV BX, AX

Rẽ nhánh (3/4)

Trong ngôn ngữ C

```
If (AL>='a' and AL<='z')
    AX = AX + 1;
else
    AX = AX – 1;
BX = AX;
```

Trong ASM (C2)

```
    CMP AL, 'a'
    JB KPTHUONG
    CMP AL, 'z'
    JA KPTHUONG
    INC AX
    JMP TIEP
KPTHUONG:
    DEC AX
TIEP:
    MOV BX, AX
```

Trong ASM (C1)

```
    CMP AL, 'a'
    JAE CTTHUONG
    DEC AX
    JMP TIEP
CTTHUONG:
    CMP AL, 'z'
    JBE THUONG
    DEC AX
    JMP TIEP
THUONG:
    INC AX
TIEP:
    MOV BX, AX
```

Trong ASM (C3)

```
    CMP AL, 'a'
    JB KPTHUONG
    CMP AL, 'z'
    JBE THUONG
KPTHUONG:
    DEC AX
    JMP TIEP
THUONG:
    INC AX
TIEP:
    MOV BX, AX
```

Rẽ nhánh (4/4)

Trong ngôn ngữ C

```
If (AL>=‘A’ and AL<=‘Z’)
        printf (“La ky tu hoa”);
else if (AL>=‘0’ and AL<=‘9’)
        printf (“La ky tu so”);
else
        printf (“La ky tu khac”);
```

Trong ASM (C2)

```
        CMP AL, ‘A'
        JB XETSO
        CMP AL, ‘Z'
        JA KHAC
        ; xuất thông báo
        ; “La ky tu hoa”
        ;
        JMP THOAT
XETSO:
        CMP AL, ‘0'
        JB KHAC
        CMP AL, ‘9'
        JA KHAC
        ; xuất thông báo
        ; “La ky tu so”
        ;
        JMP THOAT
KHAC:
        ; xuất thông báo
        ; “La ky tu khac”
        ;
THOAT:
```

Trong ASM (C1)

```
        CMP AL, 'A'
        JAE LAHOA
        CMP AL, 'Z'
        JBE LAHOA
        CMP AL, '0'
        JAE LASO
        CMP AL, '9'
        JBE LASO
KHAC:
        ; xuất thông báo
        ; “La ky tu khac”
        ;
        JMP THOAT
LAHOA:
        ; xuất thông báo
        ; “La ky tu hoa”
        ;
        JMP THOAT
LASO:
        ; xuất thông báo
        ; “La ky tu so”
        ;
THOAT:
```

SAI

Trong ASM (C3)

```
        CMP AL, '0'
        JB KHAC
        CMP AL, '9'
        JBE LASO
        CMP AL, 'A'
        JB KHAC
        CMP AL, 'Z'
        JBE LAHOA
KHAC:
        ; xuất thông báo
        ; “La ky tu khac”
        ;
        JMP THOAT
LASO:
        ; xuất thông báo
        ; “La ky tu so”
        ;
        JMP THOAT
LAHOA:
        ; xuất thông báo
        ; “La ky tu hoa”
        ;
THOAT:
```

Rẽ nhánh (4/4)

Trong ngôn ngữ C

```
If (AL>='A' and AL<='Z')
        printf ("La ky tu hoa");
else if (AL>='0' and AL<='9')
        printf ("La ky tu so");
else
        printf ("La ky tu khac");
```

Trong ASM (C2)

```
        CMP AL, 'A'
        JB XETSO
        CMP AL, 'Z'
        JA KHAC
        ; xuất thông báo
        ; "La ky tu hoa"
        ;
        JMP THOAT
XETSO:
        CMP AL, '0'
        JB KHAC
        CMP AL, '9'
        JA KHAC
        ; xuất thông báo
        ; "La ky tu so"
        ;
        JMP THOAT
KHAC:
        ; xuất thông báo
        ; "La ky tu khac"
        ;
THOAT:
```

Trong ASM (C1)

```
        CMP AL, 'A'
        JAE CTLAHOA
        JMP KHAC
CTLAHOA:
        CMP AL, 'Z'
        JBE LAHOA
        CMP AL, '0'
        JAE CTLASO
        JMP KHAC
CTLASO:
        CMP AL, '9'
        JBE LASO
        JMP KHAC
LAHOA:
        ; xuất thông báo
        ; "La ky tu hoa"
        ;
        JMP THOAT
LASO:
        ; xuất thông báo
        ; "La ky tu so"
        ;
        JMP THOAT
KHAC:
        ; xuất thông báo
        ; "La ky tu khac"
        ;
        JMP THOAT
THOAT:
```

SAI

Trong ASM (C3)

```
        CMP AL, '0'
        JB KHAC
        CMP AL, '9'
        JBE LASO
        CMP AL, 'A'
        JB KHAC
        CMP AL, 'Z'
        JBE LAHOA
KHAC:
        ; xuất thông báo
        ; "La ky tu khac"
        ;
        JMP THOAT
LASO:
        ; xuất thông báo
        ; "La ky tu so"
        ;
        JMP THOAT
LAHOA:
        ; xuất thông báo
        ; "La ky tu hoa"
        ;
THOAT:
```

Rẽ nhánh (4/4)

Trong ngôn ngữ C

```
If (AL>='A' and AL<='Z')
        printf ("La ky tu hoa");
else if (AL>='0' and AL<='9')
        printf ("La ky tu so");
else
        printf ("La ky tu khac");
```

Trong ASM (C2)

```
        CMP AL, 'A'
        JB XETSO
        CMP AL, 'Z'
        JA KHAC
        ; xuất thông báo
        ; "La ky tu hoa"
        ;
        JMP THOAT
XETSO:
        CMP AL, '0'
        JB KHAC
        CMP AL, '9'
        JA KHAC
        ; xuất thông báo
        ; "La ky tu so"
        ;
        JMP THOAT
KHAC:
        ; xuất thông báo
        ; "La ky tu khac"
        ;
THOAT:
```

Trong ASM (C1)

```
        CMP AL, '0'
        JAE CTLASO
        JMP KHAC
CTLASO:
        CMP AL, '9'
        JBE LASO
        CMP AL, 'A'
        JAE CTLAHOA
        JMP KHAC
CTLAHOA:
        CMP AL, 'Z'
        JBE LAHOA
        JMP KHAC
LASO:
        ; xuất thông báo
        ; "La ky tu so"
        ;
        JMP THOAT
LAHOA:
        ; xuất thông báo
        ; "La ky tu hoa"
        ;
        JMP THOAT
KHAC:
        ; xuất thông báo
        ; "La ky tu khac"
        ;
        JMP THOAT
THOAT:
```

Trong ASM (C3)

```
        CMP AL, '0'
        JB KHAC
        CMP AL, '9'
        JBE LASO
        CMP AL, 'A'
        JB KHAC
        CMP AL, 'Z'
        JBE LAHOA
KHAC:
        ; xuất thông báo
        ; "La ky tu khac"
        ;
        JMP THOAT
LASO:
        ; xuất thông báo
        ; "La ky tu so"
        ;
        JMP THOAT
LAHOA:
        ; xuất thông báo
        ; "La ky tu hoa"
        ;
THOAT:
```

Một vài lưu ý về rẽ nhánh

- Thường dùng cách (2) hoặc cách (3) để thực hiện rẽ nhánh
  - Cách (3) dễ hiểu đối với người đọc
  - Cách (2) hơi khó hiểu hơn nhưng ánh xạ tương ứng với mã ngôn ngữ C
- Cách (1) ít được sử dụng
  - Chỉ nên sử dụng trong trường hợp so sánh bằng
  - Khi điều kiện so sách phức tạp thì việc quản lý rẽ nhánh sẽ trở nên rất phức tạp và khó kiểm soát
- …

```
JMP TEST   ß THỪA
TEST:
…
```

…

```
J<cc> TEST   ß SAI
TEST:
…
```

Cách diễn đạt (1/2)

```
mov ax, bl ???
```

```
mov al, bl
cbw
```

hoặc

```
mov al, bl
mov ah, 0
```

```
s db 12h, 34h, 56h, 78h, 90h
mov al, s[0]
mov s[1], ax
mov s[3], 5
```

```
; al = 12
; lỗi à mov word ptr s[1], ax
; s = 12, 12, 00, 05, 90
```

```
lea si, s
mov al, [si]
mov [si+1], ax
mov [si+3], 5
```

```
; al = 12
; s = 12, 12, 00, 78, 90
; lỗi à mov byte ptr [si+3], 5
; s = 12, 12, 00, 05, 90
```

```
s dw 0123h, 0456h, 0789h ???
```

Cách diễn đạt (2/2)

Trong ngôn ngữ C	Trong ASM:
a = b;	mov AX, b mov a, AX

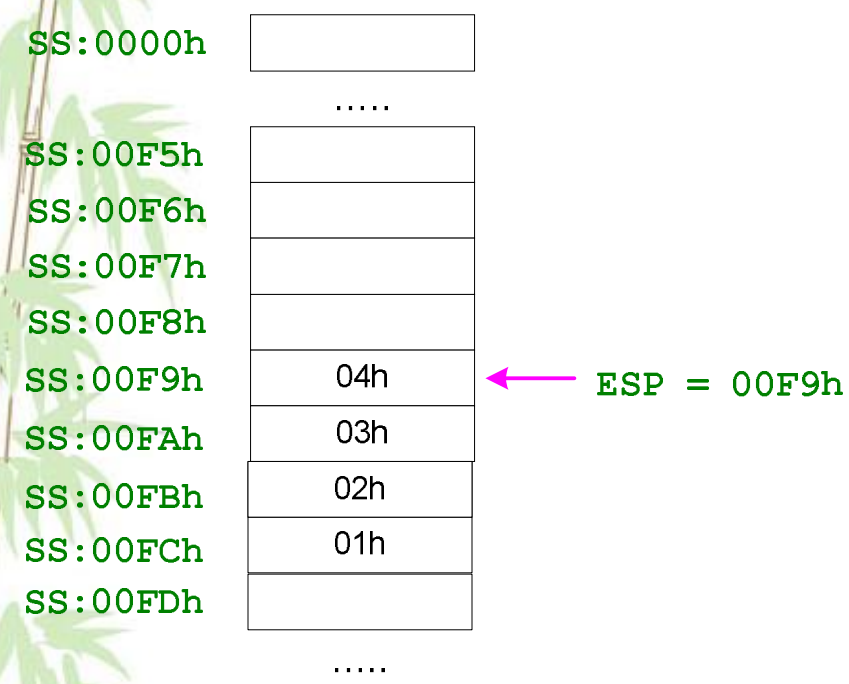
x = y*z;	mov AX, y
	imul z
	mov x, AX

Tại sao không có imul z,x,y ? mov a,b ?

Ngăn xếp (Stack)

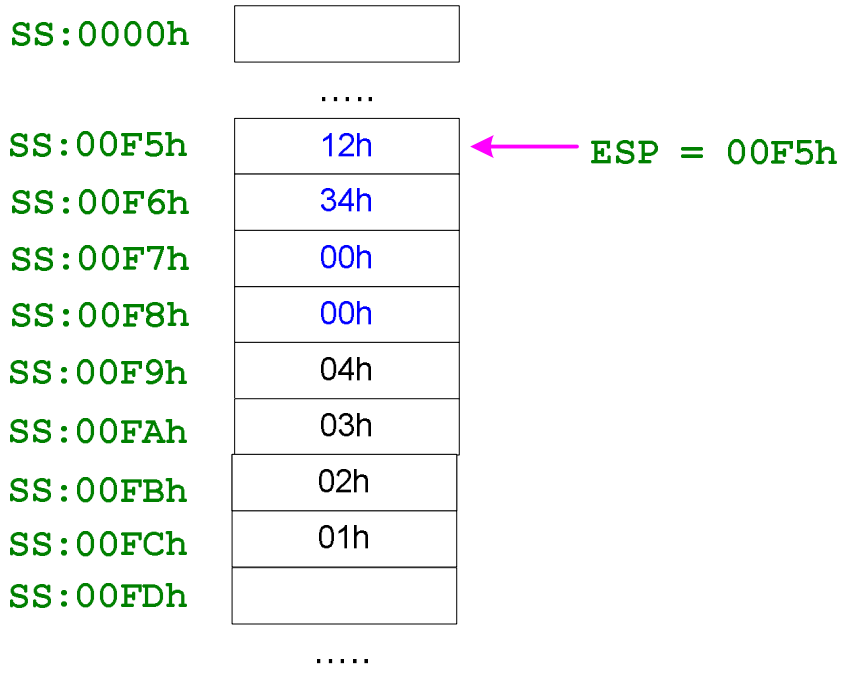
- Ngăn xếp là một vùng nhớ
  - Hoạt động theo cơ chế LIFO (Last In First Out)
  - Được sử dụng theo chiều giảm của địa chỉ (khác với các vùng nhớ thông thường được sử dụng theo chiều tăng của địa chỉ)
- Cặp thanh ghi SS:ESP chứa địa chỉ segment:offset của của đỉnh ngăn xếp
- Lệnh *PUSH <toán hạng>*
  - *Toán hạng* có thể là thanh ghi/vùng nhớ/hằng số 4 byte
  - Giảm ESP đi 4
  - Đưa giá trị của toán hạng vào ô nhớ có địa chỉ SS:ESP.
- Lệnh *POP <toán hạng>*
  - *Toán hạng* có thể là thanh ghi/vùng nhớ 4 byte
  - Đưa giá trị từ ô nhớ có địa chỉ SS:ESP vào toán hạng
  - Tăng ESP lên 4

Ví dụ lệnh PUSH



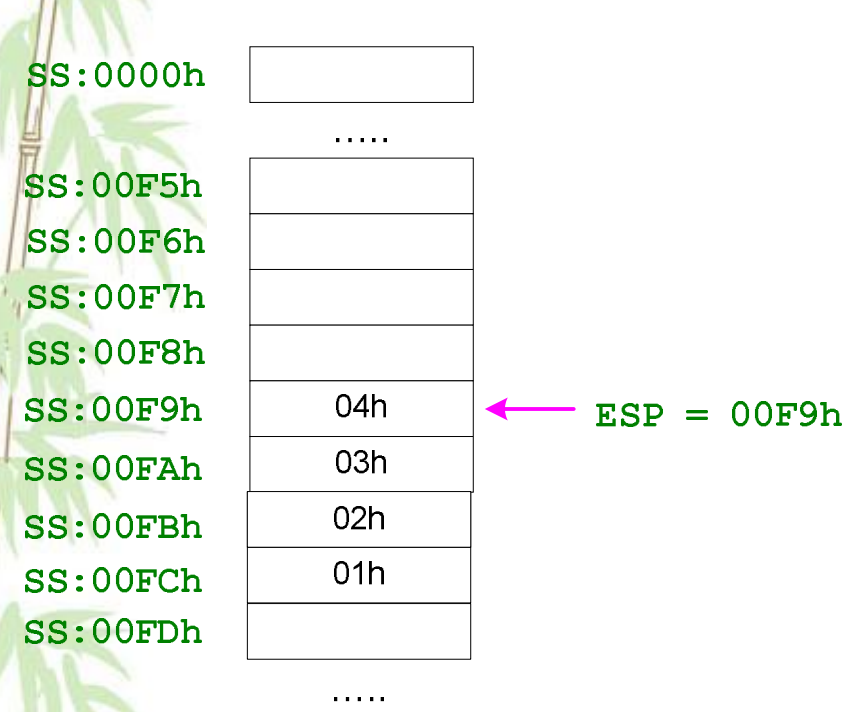
EAX có giá trị 3412h

Trước thao tác PUSH EAX

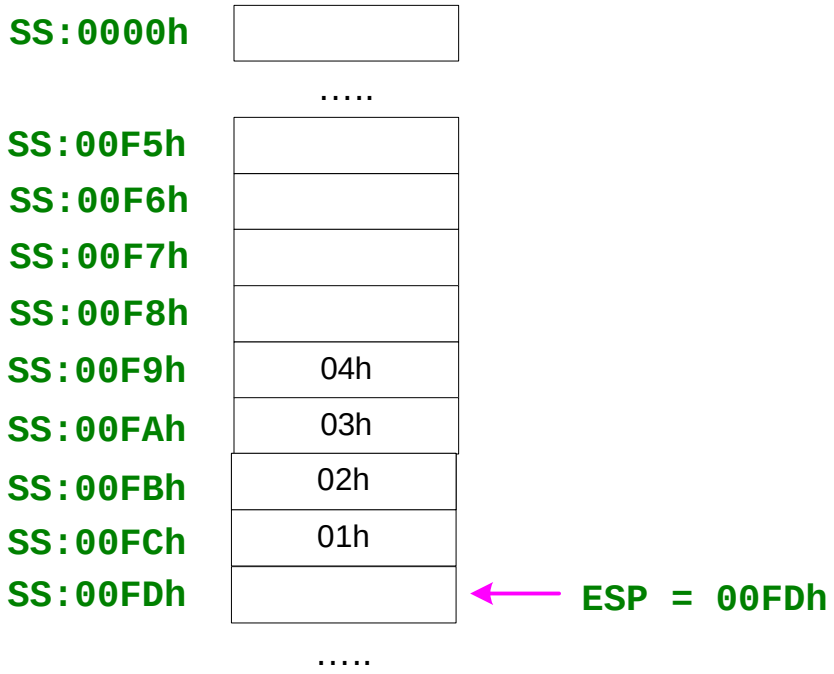


Sau thao tác PUSH EAX

Ví dụ lệnh POP



Trước thao tác POP EBX



Sau thao tác POP EBX

EBX có giá trị 01020304h

Thủ tục (Procedure)

- Lệnh *CALL <Tên thủ tục>*
  - *Sử dụng stack* để lưu trữ (PUSH) địa chỉ của *lệnh tiếp ngay sau lệnh CALL* (nơi cần quay lại)
  - Ghi vào thanh ghi con trỏ lệnh *EIP* địa chỉ của *lệnh đầu tiên của chương trình con*.
- Khai báo thủ tục

```
<Tên thủ tục> PROC
    .
    .
    ret
<Tên thủ tục> ENDP
```

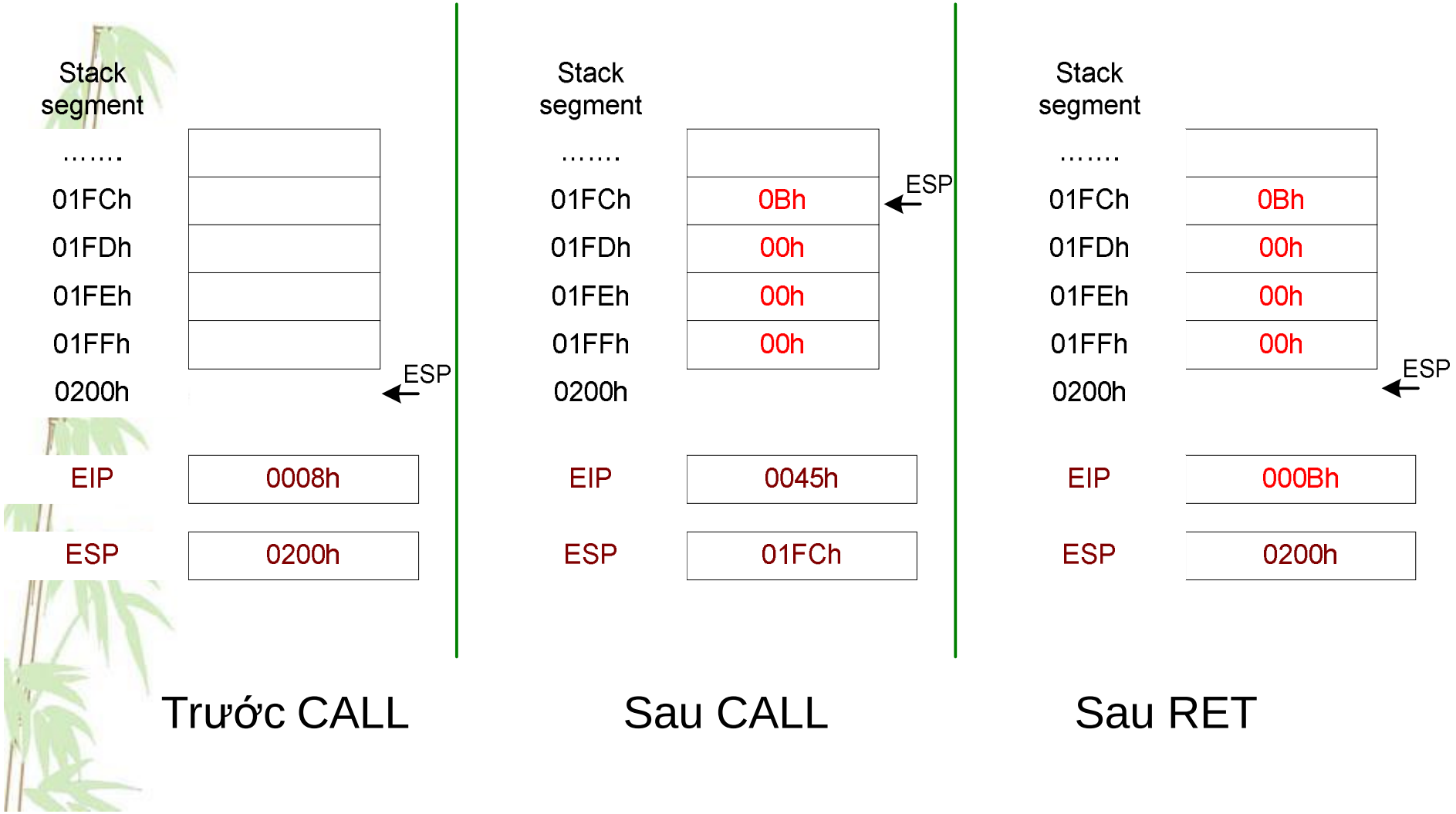
```
sample PROC
    .
    .
    ret
sample ENDP
```

- Lệnh RET
  - Lấy (POP) giá trị từ đỉnh ngăn xếp và ghi vào thanh ghi con trỏ lệnh EIP, làm cho lệnh tiếp theo được thực hiện chính là lệnh ngay sau lệnh CALL.

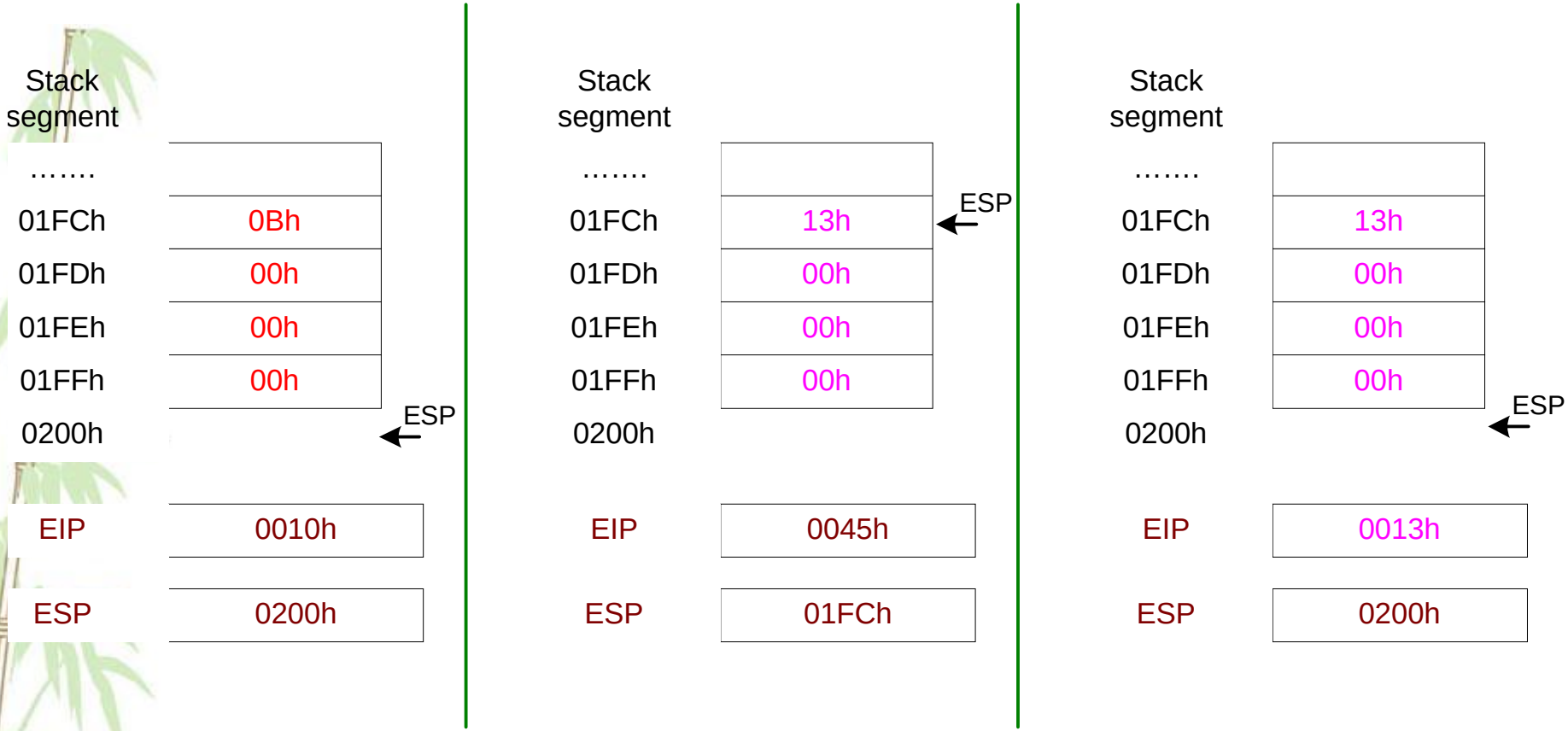
Ví dụ gọi thủ tục

```
                    section .code
                      ...
0005h                 MOV   AX,'a'
0008h                 CALL  ToUpper
000Bh                 MOV   BX,AX
000Dh                 MOV   AX,'z'
0010h                 CALL  ToUpper
0013h                 MOV   CX,AX
                      ...
                      ...
                      ...
                      ...
                    ToUpper PROC
0045h                 SUB   AX,20h
0048h                 RET
                    ToUpper ENDP
                      ...
```

Diễn giải lời gọi thủ tục *ToUpper* thứ 1



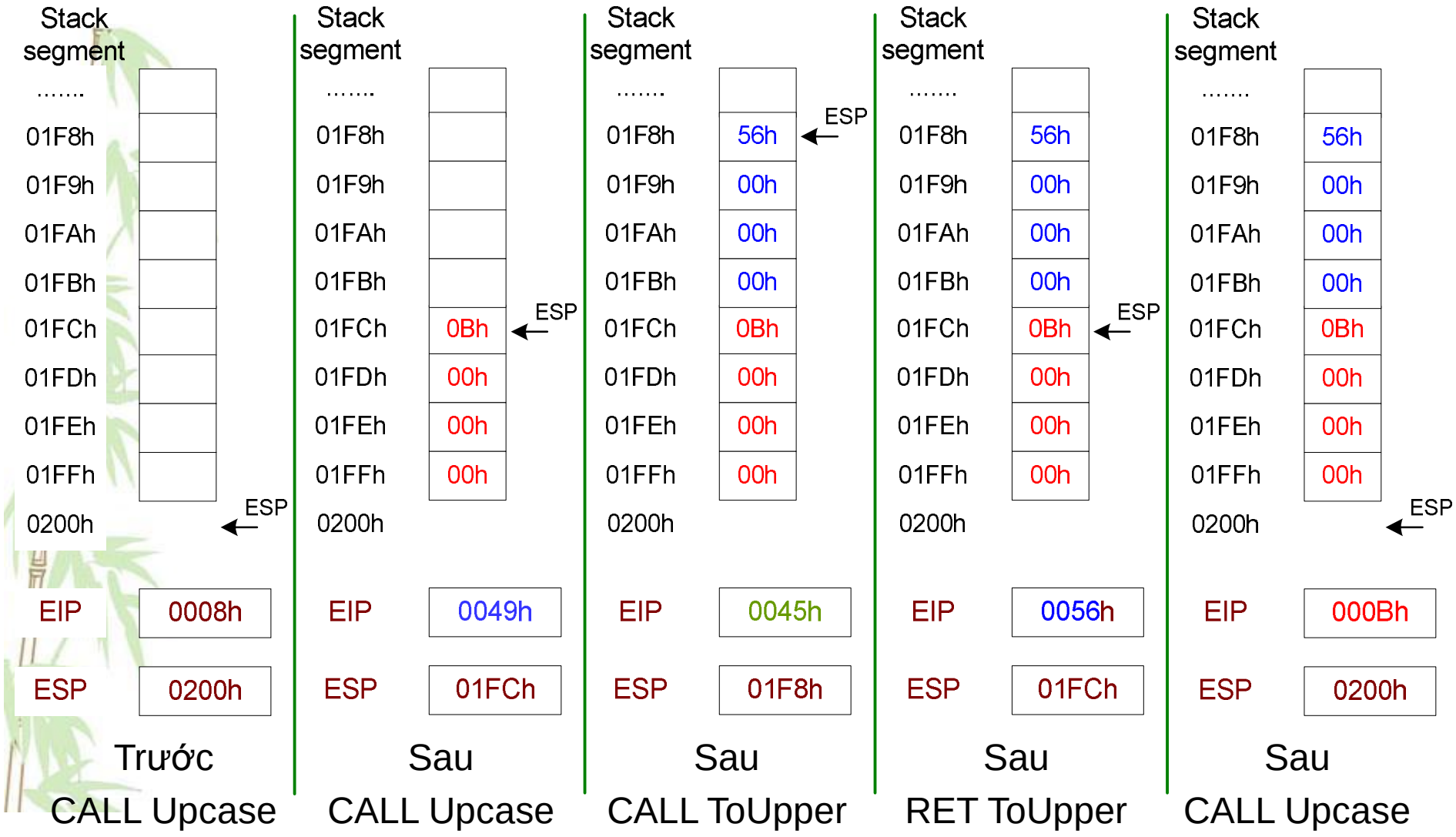
Diễn giải lời gọi thủ tục *ToUpper* thứ 2



Ví dụ gọi thủ tục lồng nhau

```
section .code
   ...
   MOV   AX,'V'          0005h
   CALL  Upcase          0008h
   MOV   BX,AX           000Bh
   MOV   AX,'n'          000Dh
   CALL  Upcase          0010h
   MOV   CX,AX           0013h
   ...
   ...
ToUpper PROC
   SUB   AX,20h          0045h
   RET                   0048h
ToUpper ENDP
Upcase PROC
   CMP   AX,'a'          0049h
   JB    Notaz           004Ch
   CMP   AX,'z'          004Eh
   JA    Notaz           0051h
   CALL  ToUpper         0053h
Notaz:
   RET                   0056h
Upcase ENDP
   ...
```

Diễn giải gọi thủ tục lồng nhau



Một số lưu ý về sử dụng Thủ tục khi lập trình hợp ngữ x86

- Khai báo thủ tục sau lời gọi thủ tục thoát
- Không nên có lệnh nhảy ra ngoài thủ tục
- Lời gọi thủ tục thao tác với ngăn xếp một cách không tường minh
- Nếu trong thân thủ tục có thao tác với ngăn xếp, nhớ lưu lại địa chỉ trả về của thủ tục
- Truyền tham số cho thủ tục
  - Thanh ghi
  - Biến toàn cục
  - **Ngăn xếp**